Software Communications Architecture Specification

APPENDIX D. DOMAIN PROFILE

Revision Summary

1.0	release for prototype implementation and validation
1.0.1	correction of XML syntax errors; deleted *deploymentattributedefinition* element (D.4.2), which was redundant with *simple* (with the addition of *action* element to *simple*) and more in line with the CORBA components spec.; deleted *deploymentattribute* (D.4.3) for same reason; changed *deploymentattributedef* element to *propertyref* (D.2.1.8.10.1) for consistency with those changes; changed "access" to "io" to be consistent with SCAS terminology; added softpkgrefid attribute to SPD and SAD to allow profile to refer to a file already loaded in radio; clarified the initial implied value of the enumeration element (D.4.1.1.6); corrected and clarified description of *ports* element in D.5.1.4.2. Added section D.7 and Attachment 1 for complete DTDs.
1.1	Incorporate approved Change Proposals, numbers 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 176, 202, 203, 212, 214, 216.
2.0	Incorporate approved Change Proposals, numbers 152, 270, 281, 308, 309, 318, 321.
2.1	Incorporate approved Change Proposals, numbers 88, 183, 306, 355, 384, 468 also complete some changes from CP 88, 142, 318, 473, 477 not incorporated in v2.0.

Change Proposals are controlled by the JTRS Change Control Board. CPs incorporated into the SCA are considered "closed" and can be seen on the JTRS web site at: www.jtrs.sarda.army.mil/docs/documents/sca_ccb.html

Table of Contents

List of Figures

APPENDIX D DOMAIN PROFILE

The Software Communications Architecture (SCA) specification provides architectural specifications for the deployment of communications software into a Software Definable Radio (SDR) device. The intent of the SDR device is to provide a re-configurable platform, which can host software components written by various vendors to support user functional services. The SCA specification requires portable software components to provide common information called a domain profile. The intent of this appendix is to clearly define to the component developers the requirements of information and format for the delivery of this information. The domain management functions use the component deployment information expressed in the Domain Profile. The information is used to start, initialize, and maintain the applications that are installed into the SCA-compliant system.

The Object Management Group (OMG) is a standards organization supporting the definition of specifications for distributed computing environments. The Object Management Group has recently created the CORBA Components Specification that defines a process for deployment of software components into an object-oriented framework. The format used by the CORBA Components Specification for deployment is the eXtensible Markup Language (XML).

This specification has been designed to follow the philosophy of the CORBA Components Model (OMG TC Document orbos/07-01-99: Chapter 10 - Packaging and Deployment). Due to the differences between the SCA Core Framework IDL and the CORBA Components Specification IDL, it was necessary to modify some of the deployment principles for use in the SCA. This specification defines the XML Document Type Definition (DTD) set for use in deploying SCA components. The complete DTD set is contained in Attachment 1 to this Appendix.

D.1 DEPLOYMENT OVERVIEW.

The hardware devices and software components that make up an SCA system domain are described by a set of files that are collectively referred to as a Domain Profile. A Domain Profile contains a set of Software Profiles. A Software Profile is either a Software Assembly Descriptor (for applications) or a Software Package Descriptor (for all other software components and hardware devices). These files describe the identity, capabilities, properties, and inter-dependencies of the hardware devices and software components that make up the system. All of the descriptive data about a system is expressed in the XML vocabulary. For purposes of this SCA specification, the elements of the XML vocabulary have been based upon the OMG's CORBA Components specification (orbos/99-07-01). (Note: At the time of this writing, 99-07-01 is a draft standard).

Figure D-1 depicts the relationships between the XML files that are used to describe a system's hardware and software assets (those describing hardware are colored green; Properties Files apply to both software and hardware and is colored blue). The XML vocabulary within each of these files describes a distinct aspect of the hardware and software assets.

A Software Assembly Descriptor file describes how multiple components of an assembly, i.e., an application, are deployed and interconnected. A Software Assembly Descriptor file is associated with one or more Software Package Descriptor files. Each component of the Software Assembly

Descriptor is described in a Software Package Descriptor file. Information about the interfaces that a component publishes and/or consumes are contained in a Software Component Descriptor file. Each Software Component Descriptor file is associated with a Software Package Descriptor file that describes one or more implementations of the software component. Software properties are described in a Properties File that may be applicable to all implementations of the component, i.e., associated at the Software Package Descriptor level or applicable to a single implementation of the component.

Three types of files, a Device Package Descriptor, a Properties File, and a Device Configuration Descriptor describe hardware devices and are known collectively as a Device Profile. Device Profiles are part of a logical CF *Device's* Software Profile. A Device Package Descriptor file identifies a class of a device. Properties Files are associated with Device Package Descriptors. The Properties File contains information about the properties of a device such as serial number, processor type, OS type and allocation capacities.

A Device Configuration Descriptor file describes how many components are initially started up on the device, and how to find the CF *DomainManager*. Child CF *Devices* will have a null Device Configuration Descriptor file when not a parent CF *Device*. Each component of the Device Configuration Descriptor is described in a Software Package Descriptor file.

A *DomainManager* Configuration Descriptor file contains configuration information for the CF *DomainManager*. A Software Package Descriptor file can describe the CF *DomainManager's* implementation in the *DomainManager* Configuration Descriptor file.

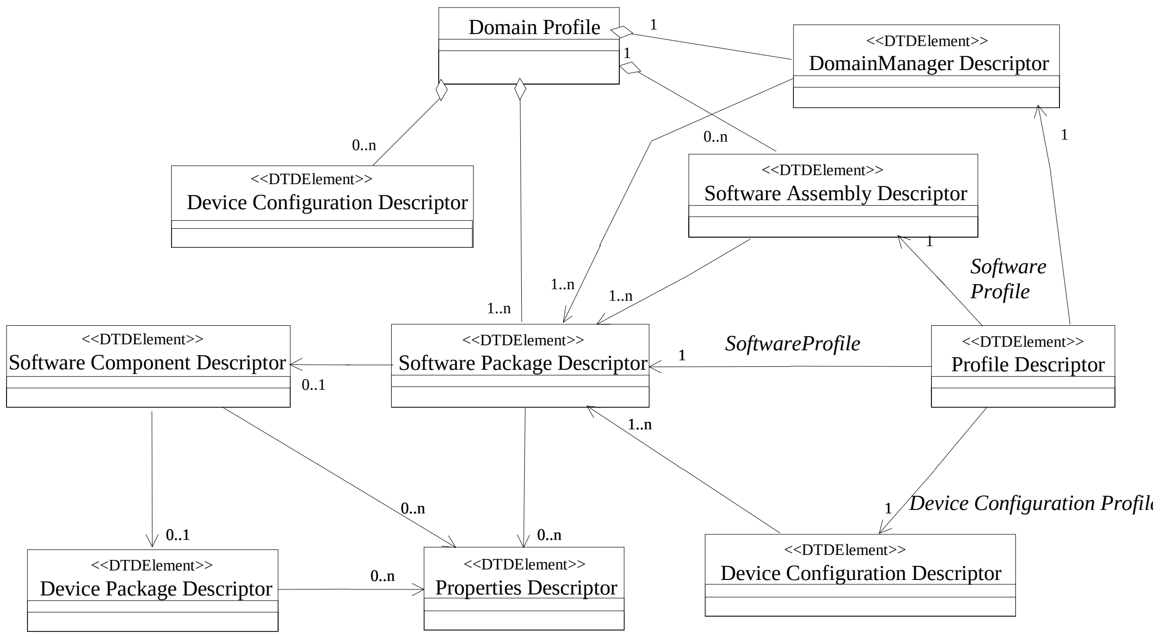

Figure D-1. Relationships Between Domain Profile XML File Types

D.2 SOFTWARE PACKAGE DESCRIPTOR.

The SCA Software Package Descriptor is used at deployment time to load an SCA compliant component and its various implementations. The information contained in the Software Package Descriptor, as shown in Figure D-2, will provide the basis for the Domain Management function to manage the component within the SCA architecture.

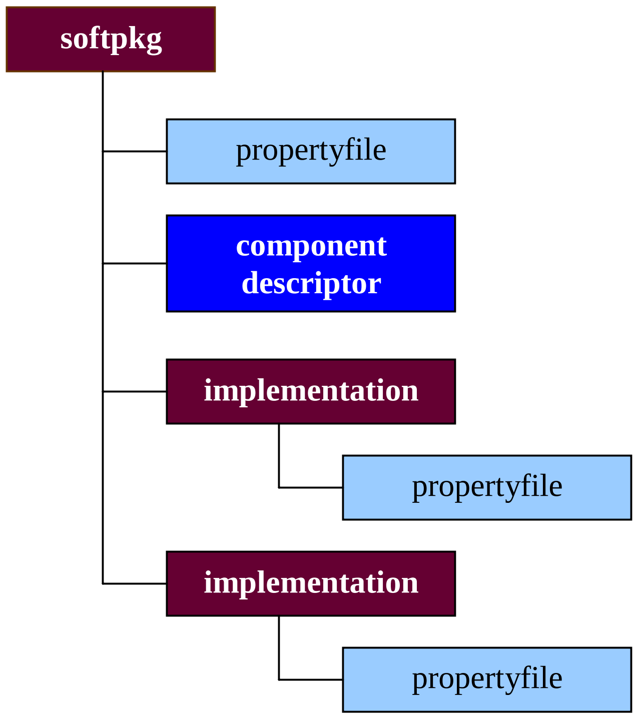

Figure D-2. Software Package Descriptor Overview

Figure D-2 details the various elements of the XML file that is to be delivered to the CF *DomainManager* during the installation of every component. The software package descriptor may contain various implementations of any given component. Within the specification of a software package descriptor several other files are referenced including a component level *propertyfile* and a software component descriptor file. Within any given implementation there may be additional *propertyfiles*.

D.2.1 Software Package.

The *softpkg* element indicates a Software Package Descriptor (SPD) definition. The *softpkg* id uniquely identifies the package and is a DCE UUID. The DCE UUID is as defined by the DCE UUID standard (adopted by CORBA). The DCE UUID format starts with the characters "DCE:" and is followed by the printable form of the UUID, a colon, and a decimal minor version number, for example: "DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1". The decimal minor version number is optional. The version specifies the version of the component; name is a user-friendly label for the *softpkg*. The type attribute indicates the type of component implementation. All files referenced by the software package are located in the same directory as the SPD file.

The set of properties for a software package may come from several sources. The properties to be used for a software package are the union of properties in the following precedence order:

1. SPD Implementation Properties -
2. SPD level properties
3. SCD properties

Any duplicate properties having the same ID are ignored. Duplicated properties must be the same property type, only the value can be over-ridden. The implementation properties are only used for the initial configuration and creation of a component by the CF *ApplicationFactory* and cannot be referenced by a SAD componentinstantiation *componentproperties* or *resourcefactoryproperties* element.

```
<!ELEMENT softpkg
       ( title?
       , author+
       , description?
       , propertyfile?
       , descriptor?
       , implementation+
       , usesdevice*
       )>
<!ATTLIST softpkg
id            ID            #REQUIRED
name          CDATA         #REQUIRED
type    (sca_compliant | sca_non_compliant)
                             "sca_compliant"
version       CDATA         #IMPLIED >
```

D.2.1.1 *propertyfile*.

The *propertyfile* element is used to indicate the local filename of the property file associated with the software package. The intent of the *propertyfile* will be to provide the definition of *properties* elements common to all implementations of the component being deployed in the software package (*softpkg*).

Property files may also contain *properties* elements that are used in definition of command and control id value pairs used by the SCA *Resource* configure( ) and query( ) interfaces. The format of the property file is described in the Properties Descriptor (Section D.4).

```
<!ELEMENT propertyfile
      (localfile
      )>
<!ATTLIST propertyfile
      type   CDATA  #IMPLIED>
```

D.2.1.1.1 *localfile.*

The *localfile* element is used to reference a file in the same directory as this XML file or a directory that is relative to the directory where this XML file is located. When the name attribute is a simple name then the file exists in the same directory as this XML file. A relative directory indication begins either with "../" meaning parent directory and "./" means current directory in the name attribute. Multiple "../" and directory names can follow the initial "../" in the name attribute. All name attributes must have a simple name at the end of the file name.

```
<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
      name   CDATA  #REQUIRED>
```

D.2.1.2 *title.*

The *title* element is used for indicating a title for the software component being installed by *softpkg*.

```
<!ELEMENT title (#PCDATA)>
```

D.2.1.3 *author.*

The *author* element will be used to indicate the name of the person, the company, and the web page of the developer producing the component being installed into the system.

```
<!ELEMENT author
      ( name*
      | company?
      | webpage?
)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT webpage (#PCDATA)>
```

D.2.1.4 *description.*

The *description* element will be used to describe any pertinent information about the software component being delivered to the system.

```
<!ELEMENT description (#PCDATA)>
```

D.2.1.5 *descriptor.*

The *descriptor* element points to the local filename of the *software component descriptor file* that is used to document the interface information for the component being delivered to the system. In the case of the *SCA Component*, the descriptor will contain information about three aspects of the component (the component type, message ports, and IDL interfaces). The software component descriptor file is optional, since some SCA components are non-CORBA components like digital signal processor (DSP) "c" code (see section on software component descriptor file, section D.5).

```
<!ELEMENT descriptor
      (localfile
      )>
<!ATTLIST descriptor
      name         CDATA  #IMPLIED>
```

D.2.1.6 *implementation.*

The *implementation* element contains descriptive information about the particular implementation template for the software component contained in the *softpkg*. The implementation id uniquely identifies the implementation and is a DCE UUID value as stated section D.2.1. The *implementation* is intended to allow multiple component templates to be delivered to the system in one software package. Each *implementation* is intended to allow the same component to support different types of processors, operating systems, etc. The *implementation* element will also allow definition of implementation-dependent properties for use in CF *Device*, CF *Application*, or CF *Resource* creation. The id element is intended to provide a universal unique identifier for the specific implementation of the component. The *compiler*, *programminglanguage*, *humanlanguage*, *os*, *processor*, and runtime elements are optional *dependency* elements. Additional *dependency* elements can be stated in the *dependency* element.

```
<!ELEMENT  implementation
       ( description?
       , propertyfile?
       , code
       , compiler?
       , programminglanguage?
       , humanlanguage?
       , runtime?
       , ( os
         | processor
         | dependency
         )+
       , usesdevice*
       )>
<!ATTLIST implementation
       id      ID      #REQUIRED
       aepcompliance (aep_compliant | aep_non_compliant) “aep_compliant”>
```

D.2.1.6.1 *propertyfile.*

The *propertyfile* element is used to indicate the local filename of the property file associated with this component package. Although the specification does not restrict the specific use of the property file based on context, it is intended within the *implementation* element to provide component implementation specific *properties* elements for use in command and control id value pair settings to the CF *Resource* configure( ) and query( ) interfaces. See the descriptions of the property file formats in the Properties Descriptor, section D.4

```
<!ELEMENT propertyfile
       (localfile
       )>
<!ATTLIST code propertyfile
       type   CDATA  #IMPLIED>

<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
       name   CDATA  #REQUIRED>
```

D.2.1.6.2 *description.*

The *description* element will be used to describe any pertinent information about the software component implementation that the software developer wishes to document within the software package profile.

```
<!ELEMENT description (#PCDATA)>
```

D.2.1.6.3 *code.*

The *code* element will be used to indicate the local filename of the code being delivered by the *softpkg* for this implementation of the software. The stacksize and priority are options parameters for the CF *ExecutableDevice execute* operation. These values are unsigned long. The *type* attribute for the *code* element will also indicate the type of file being delivered to the system. The *entrypoint* element provides the means for providing the name of the entry point of the component being delivered. The valid values for the type attribute are: “Executable”, “KernelModule”, “SharedLibrary”, and “Driver.”

The meaning of the code type attribute:

1. Executable means to use CF *LoadableDevice load* and CF *ExecutableDevice execute*. This is a “main” process.
2. Driver and Kernel Module means load only.
3. SharedLibrary means dynamic linking.
   a. Without a code *entrypoint* element means load only.
   b. With a code *entrypoint* element means load and CF *Device execute*.

```
<!ELEMENT code
```

```
            ( localfile
            , entrypoint?
            , stacksize?
            , priority?
            )>
    <!ATTLIST code
            type   CDATA  #IMPLIED>

    <!ELEMENT localfile EMPTY>
    <!ATTLIST localfile
            name   CDATA  #REQUIRED>

    <!ELEMENT entrypoint (#PCDATA)>

    <!ELEMENT stacksize (#PCDATA)>

    <!ELEMENT priority (#PCDATA)>
```

D.2.1.6.4 compiler.

The *compiler* element will be used to indicate the compiler used to build the software component being delivered by *softpkg*. The required *name* attribute will specify the name of the compiler used, and the *version* attribute will contain the compiler version.

```
    <!ELEMENT compiler EMPTY>
    <!ATTLIST compiler
            name    CDATA #REQUIRED
            version CDATA #IMPLIED>
```

D.2.1.6.5 *programminglanguage.*

The *programminglanguage* element will be used to indicate the type of programming language used to build the component implementation. The required *name* attribute will specify a language such as “c”, “c++”, and “java”..

```
    <!ELEMENT programminglanguage EMPTY>
    <!ATTLIST programminglanguage
            name    CDATA #REQUIRED
            version CDATA #IMPLIED>
```

D.2.1.6.6 *humanlanguage.*

The *humanlanguage* element will be used to indicate the human language the component is developed for.

```
    <!ELEMENT humanlanguage EMPTY>
    <!ATTLIST humanlanguage
            name CDATA #REQUIRED>
```

D.2.1.6.7 *os.*

The *os* element will be used to indicate the *operating system* that the software component is built to operate on. The required *name* attribute will indicate the name of the operating system and the *version* attribute will contain the operating system. The *os* attributes will be defined in a property file as an allocation property of string type and with names os_name and *os*_version and with an *action* element value other than “external”. The *os* element is automatically interpreted as a dependency and compared against allocation properties with names of *os*_name and *os*_version. Legal *os*_name attribute values are listed in Attachment 2 to this appendix.

```
<!ELEMENT os EMPTY>
<!ATTLIST os
      name CDATA #REQUIRED
      version CDATA #IMPLIED>
```

D.2.1.6.8 *processor.*

The *processor* element will be used to indicate the *processor* and/or *processor family* that this software component is built to operate on.. The processor name attribute will be defined in a property file as an allocation property of string type and with a name of processor_name and with an *action* element value other than “external”. The *processor* element is automatically interpreted as a dependency and compared against an allocation property with a name of processor_name. Legal processor_name attribute values are listed in Attachment 2 to this appendix.

```
<!ELEMENT processor EMPTY><!ATTLIST processor
      name   CDATA  #REQUIRED>
```

D.2.1.6.9 *dependency.*

The *dependency* element is used to indicate the dependent relationships between the components being delivered and other components and devices in the SCA compliant system. The *softpkgref* element is used to specify a Software Package file that must be loaded before this component is loaded in the system in order for the component to load without errors. The propertyref will reference a specific allocation property by a unique identifier, and provides the value that will be used against a CF *Device* capacity model. The CF *DomainManager* will use these dependency definitions to assure that components and devices that are necessary for proper operation of the implementation are present and available. The type attribute is descriptive information indicating the type of dependency.

```
<!ELEMENT dependency
      ( softpkgref
      | propertyref
      )>
<!ATTLIST dependency
      type      CDATA      #REQUIRED>
```

D.2.1.6.9.1 *softpkgref.*

The *softpkgref* element refers to an external *softpkg*. The *softpkgref* element indicates a load dependency. The file is referenced by *localfile* element. An optional *implref* element refers to a particular implementation unique identifier within the *softpkg* descriptor.

```
<!ELEMENT softpkgref
      ( localfile
      , implref?
      )>

<!ELEMENT implref EMPTY>
<!ATTLIST implref
      refid  CDATA  #REQUIRED>
```

D.2.1.6.9.2 *propertyref.*

The *propertyref* element is used to indicate a unique id that references a defined *simple* allocation property in the package, and a propertyvalue used by the Domain Management function to perform the dependency check. This id is a DCE UUID value as specified in section D.2.1.

```
<!ELEMENT propertyref EMPTY>
<!ATTLIST propertyref
      refid        CDATA  #REQUIRED
      value        CDATA  #REQUIRED>
```

D.2.1.6.10 runtime.

The *runtime* element specifies a runtime required by a component implementation. An example of the runtime is a Java VM.

```
<!ELEMENT runtime EMPTY>
<!ATTLIST runtime
      name         CDATA  #REQUIRED>
      version      CDATA #IMPLIED>
```

D.2.1.7 *usesdevice.*

The *usesdevice* will be used to describe any uses relationships this component has with another device in the system. The *propertyref* references allocation properties to indicate the CF *Device* to be used and/or the capacity needed from the CF *Device*.

```
<!ELEMENT  usesdevice
      ( propertyref+
      )>
<!ATTLIST usesdevice
      id     ID          #REQUIRED
      type   CDATA  #REQUIRED>
```

D.2.1.7.1 *propertyref.*

See D.2.1.6.9.2 for a definition of the propertyref element.

D.3 DEVICE PACKAGE DESCRIPTOR.

The SCA Device Package Descriptor is the part of a Device Profile that contains hardware device Registration attributes, which are typically used by a Human Computer Interface application to display information about the device(s) resident in an SCA-compliant radio system. DPD information is intended to provide hardware configuration and revision information to a radio operator or to radio maintenance personnel. A DPD may be used to describe a single hardware element residing in a radio or it may be used to describe the complete hardware structure of a radio. In either case, the description of the hardware structure should be consistent with hardware partitioning as described in the Hardware Architecture Definition in section 4.0 of the SCA.

D.3.1 Device Package.

The *devicepkg* element is the root element of the DPD. The *devicepkg* id attribute uniquely identifies the package and is a DCE UUID, as defined in paragraph D.2.1. The version attribute specifies the version of the *devicepkg*. The format of the version string is numerical major and minor version numbers separated by commas (e.g., "1,0,0,0"). The name attribute is a user-friendly label for the *devicepkg*.

```
<!ELEMENT devicepkg
     ( title?
     , author+
     , description?
     , hwdeviceregistration
     )>
<!ATTLIST devicepkg
     id        ID        #REQUIRED
     name      CDATA          #REQUIRED
     version   CDATA          #IMPLIED>
```

D.3.1.1 *title*.

The *title* element is used for indicating a title for the hardware device being described by *devicepkg*.

```
<!ELEMENT title (#PCDATA)>
```

D.3.1.2 *author*.

See D.1.1.1 for a definition of the *author* element.

D.3.1.3 *description*.

The *description* element is used to describe any pertinent information about the device implementation that the hardware developer wishes to document within the Device Package.

```
<!ELEMENT description (#PCDATA)>
```

D.3.1.4 *hwdeviceregistration*.

The *hwdeviceregistration* element provides device-specific information for a hardware device. The *hwdeviceregistration* id attribute uniquely identifies the device and is a DCE UUID, as defined in paragraph D.2.1. The version attribute specifies the version of the *hwdeviceregistration* element. The format of the version string is numerical major and minor version numbers separated by commas (e.g., "1,0,0,0"). The name attribute is a user-friendlylabel for the hardware device being registered. At a minimum, the *hwdeviceregistration* element must include a description, the manufacturer, the model number and the device's hardware class(es) (Refer to SCA section 4, Hardware Architecture Definition).

```
<!ELEMENT hwdeviceregistration
      ( propertyfile?
      , description
      , manufacturer
      , modelnumber
      , deviceclass
      , childhwdevice*
      )>

<!ATTLIST hwdeviceregistration
     id        ID            #REQUIRED
     name      CDATA         #REQUIRED
     version   CDATA         #IMPLIED>
```

D.3.1.4.1 *propertyfile*.

The *propertyfile* element is used to indicate the local filename of the property file associated with the *hwdeviceregistration* element. The format of a property file is described in the Properties Descriptor (Section D.4).

The intent of the property file is to provide the definition of properties elements for the hardware device being deployed and described in the Device Package (*devicepkg*) or *hwdeviceregistration* element.

```
<!ELEMENT propertyfile
      ( localfile
      )>
<!ATTLIST propertyfile
      type     CDATA     #IMPLIED>

<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
      name   CDATA     #REQUIRED>
```

D.3.1.4.2 *description*.

See D.2.1.4 for definition of the *description* element.

D.3.1.4.3 *manufacturer*.

The *manufacturer* element is used to convey the name of manufacturer of the device being installed.

```
<!ELEMENT manufacturer (#PCDATA)>
```

D.3.1.4.4 *modelnumber*.
The *modelnumber* element is used to indicate the manufacture's model number, for the device being installed.

```
<!ELEMENT modelnumber (#PCDATA)>
```

D.3.1.4.5 *deviceclass*.
The *deviceclass* element is used to identify one or more hardware classes that make up the device being installed (e.g., RF, Modem, I/O, as defined in SCA section 4.2.2 HWModule(s) Class Structure).

```
<!ELEMENT deviceclass
      ( class+
      )>
<!ELEMENT class (#PCDATA)>
```

D.3.1.4.6 *childhwdevice*.
The *childhwdevice* element indicates additional device-specific information for hardware devices that make up the root or parent hardware device registration. An example of *childhwdevice* would be a radio's RF module that has receiver and exciter functions within it. In this case, a CF *Device* representing the RF module itself would be a parent *Device* with its DPD, and the receiver and exciter are child devices to the module. The parent / child relationship indicates that when the RF module is removed from the system, the receiver and exciter devices are also removed.

```
<!ELEMENT childhwdevice
      ( hwdeviceregistration
      | devicepkgref
      )>
```

D.3.1.4.7 *hwdeviceregistration*.
The *hwdeviceregistration* element provides device-specific information for the child hardware device. See D.3.1.4 for definition of the *hwdeviceregistration* element.

D.3.1.4.8 *devicepkgref*.
The *devicepkgref* element is used to indicate the local filename of a Device Package Descriptor file pointed to by Device Package Descriptor (e.g., a devicepkg within a devicepkg).

```
<!ELEMENT devicepkgref
      ( localfile
      )>
<!ATTLIST devicepkgref
      type      CDATA      #IMPLIED>
```

D.4 PROPERTIES DESCRIPTOR.

The property file details component, device or home attribute settings. For purposes of the SCA, property files will contain *simple*, *simplesequence*, *test*, *struct* and *structsequence* elements. The *properties* element will be used to describe attributes of a component that will be used for *dependency* checking. The *properties* element will also be used for SCA component values of the *configure*( ), *query*( ), and *runTest*( ) operations of the CF *Resource* component.

D.4.1 *properties*.

The *properties* element is used to describe property attributes that will be used in the *configure*( ) and *query*( ) operations for SCA CF *Resource* components and for definition of attributes used for dependency checking. Properties can also used in the CF *TestableObject runTest*( ) operation to configure tests and provide test results.

```
<!ELEMENT properties
      ( description?
       ,     ( simple
             | simplesequence
                   | test
             | struct
             | structsequence
             )+
      )>
```

D.4.1.1 *simple.*

The *simple* element provides for the definition of a property which includes a unique id, name, type, and value of the property attribute that will be used in the CF *Resource configure*( ) and *query*( ) operations, for indication of component capabilities, or in the CF *TestableObject runTest* operation. The *simple* element is specifically designed to support id-value pair definitions. A *simple* property id attribute corresponds to the id of the id-value pair and the value, and range of a simple property correspond to the value of the id-value pair. If no value is given then the property cannot be used for input test values for testing and/or as an initial configuration or execute parameter of a component. The optional *enumerations* element allows for the definition of a label to value for a particular property. The *mode* element defines whether the *properties* element is readonly, writeonly or readwrite. The *id* element is an identifier for the properties. The id for a *simple* property that is an allocation type is a DCE UUID value as specified in section D.2.1. The id for all other *simple* properties can be any valid XML ID type. The mode attribute is only meaningful when the kind element is configured.

```
<!ELEMENT simple
      ( description?
      , value?
      , units?
      , range?
      , enumerations?
      , kind*
      , action?
      )>
```

```
<!ATTLIST simple
      id     ID                              #REQUIRED
      type  ( boolean | char  | double | float
             | short   | long  | objref | octet
             | string | ulong  | ushort )     #REQUIRED
      name   CDATA                           #IMPLIED
      mode  ( readonly | readwrite | writeonly)  “readwrite”>
```

D.4.1.1.1 *description.*
The *description* element is used to provide a description of the *properties* element that is being defined.

```
<!ELEMENT description (#PCDATA)>
```

D.4.1.1.2 *value.*
The *value* element is used to provide a value setting to the *properties* element.

```
<!ELEMENT value (#PCDATA)>
```

D.4.1.1.3 *units.*
The *units* element describes the intended practical data representation to be used for the *properties* element.

```
<!ELEMENT units (#PCDATA)>
```

D.4.1.1.4 *range.*
The *range* element describes the specific *min* and *max* values that are legal for the *simple* element. The intent of the *range* element is to provide a means to perform range validation. This element is not used by the CF *ApplicationFactory* or CF *Application* implementations.

```
<!ELEMENT range EMPTY
<!ATTLIST range
      min    CDATA  #REQUIRED
      max    CDATA  #REQUIRED>
```

D.4.1.1.5 *enumerations.*
The *enumerations* element is used to specify one or more *enumeration* elements.

```
<!ELEMENT enumerations
      ( enumeration+
      )>
```

The *enumeration* element is used to associate values for the given *property* attribute a label to each specific value of the property attribute. Enumerations are legal for specific *properties* elements of various integer types. An Enumeration value is assigned to a property that implements the CORBA long type. Enumeration values are implied; if not specified by a developer, the initial implied value is 0 and subsequent values are incremented by 1.

Note: The advantage of the *enumeration* element over the *sequence* element from the CORBA components specification is that the *enumeration* element provides a mechanism to associate a value of a property to a label. The *sequence* element of the CORBA component specification does not allow association of values (only lists of sequences).

```
<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
      label  CDATA  #REQUIRED
      value  CDATA  #IMPLIED>
```

D.4.1.1.6 *kind.*

The *kind* element is used to specify the kind of property. The types of *kinds* are:

1. configure which is used in the *configure*( ) and *query* ( ) operations of the CF *Resource* interface. The CF *ApplicationFactory* will use these properties to build the CF *Properties* input parameter to the *configure* ( ) operation that is invoked on the CF *Resource* components during application creation. The CF *ApplicationFactory* will also use these properties for CF *ResourceFactory create* options parameters. When the mode is readonly, only the query behavior is supported. When the mode is writeonly, only the *configure* behavior is supported. When the mode is readwrite, both *configure* and *query* are supported.

2. test which is used in the *runTest*( ) operation in the CF *TestableObject* interface. The CF *ApplicationFactory* will use these properties as the unsigned long input parameter to the *runTest*( ) operation that is invoked on the CF *Resource* components during application creation. Therefore, any *simple* element that has “test” specified in the *kind* element must have a type attribute of “ulong”.

3. allocation which is used in the *allocateCapacity*( ) and *deallocateCapacity*( ) operations of the CF *Device* interface. The CF *DeviceManager* will use these properties to build the CF *DataType* inout parameter to the *allocateCapacity*( ) operation that is invoked on the CF *Device* components during application creation. Allocation properties that are external can also be queried using the CF *PropertySet query* operation.

4. execparam which is used in the *execute* operations of the CF *Device* interface. The CF *DeviceManager* will use these properties to build the CF *Properties* input parameter to the *execute*( ) or *excuteProcess*( ) operation that is invoked on the CF *Device* components during CF *Device* and/or CF *Application* creation. Only simple elements can be used as execparam types.

5. factoryparam which is used in the createResource operations of the CF *ResourceFactory* interface. The CF *ApplicationFactory* will use these properties to build the CF *Properties* input parameter to the *createResource*( ) operation.

A property can have multiple *kinds* and the default is configure.

```
<!ELEMENT kind EMPTY>
<!ATTLIST kind
      kindtype  ( allocation | configure | test |
                  execparam | factoryparam) “configure”>
```

D.4.1.1.7 *action.*

The *action* element is used to describe the relationship used in comparison of the property attribute during the process of checking SPD dependencies. The type attribute of the *simple* element will determine the type of comparison to be done when checking the dependency against the property that has been defined.

In principle the *action* defines the operation executed during the comparison of the allocation property value provided by the SPD dependency element against the associated allocation property value of a CF *Device*. The allocation property is on the left side of the action and the dependency value is on the right side of the action. This process allows for the allocation of appropriate objects within the system based on their attributes as defined by the dependent relationships.

For example, if a CF *Device's* properties file defines an allocation DeviceKind property where the *action* element is set to "equal", then at the time of dependency checking a valid DeviceKind property is checked for equality. If a software component implementation is dependent on a DeviceKind property with a value set to "NarrowBand", then the component's SPD dependency *propertyref* will reference the id of the allocation DeviceKind property with a value of "NarrowBand". At time of dependency checking the CF *ApplicationFactory* will check CF *Devices* that have a DeviceKind allocation property for equality against a "NarrowBand" value.

```
<!ELEMENT action EMPTY>
<ATTLIST action
      type  ( eq  |  ne  | gt  | lt |
              ge  |  le  | external )  "external">
```

D.4.1.2 *simplesequence.*

The *simplesequence* element is used to specify a list of *properties* with the same characteristics (e.g., type, range, units, etc.). This definition is similar to the *simple* element except for a *simplesequence* element has a list of values instead of one value. The *simplesequence* element maps to the sequence types defined in the CF and PortTypes CORBA modules based upon the type element.

```
<!ELEMENT simplesequence
      ( description?
      , values?
      , units?
      , range?
      , kind*
      , action?
      )>
<!ATTLIST simplesequence
      id    ID                                     #REQUIRED
      type   ( boolean | char  | double | float
             | short | long | objref | octet
             | string | ulong |ushort )       #REQUIRED
      name   CDATA                                #IMPLIED
      mode   (readonly | readwrite | writeonly) “readwrite”>

<!ELEMENT values
      (value+
      )>
```

D.4.1.3 *test.*

The *test* element is used to specify a list of test properties for executing a component specific test. This definition contains *inputvalue* and *resultvalue* elements and it has a testid attribute for grouping test properties to a specific test. *Inputvalues* are used to configure the test to be performed (e.g., frequency and RF power output level). When the test has completed, *resultvalues* contain the results of the testing (e.g., Pass or a fault code/message).

```
      <!ELEMENT test
            ( description
            , inputvalue?
            , resultvalue
            )>

      <!ATTLIST test
            id     ID     #REQUIRED>
```

D.4.1.3.1 *inputvalue*.

The *inputvalue* element is used to provide test configuration properties.

```
      <!ELEMENT inputvalue
            ( simple+
            )>
```

D.4.1.3.2 *resultvalue*.

The *resultvalue* element is used to provide test result properties.

```
      <!ELEMENT resultvalue
            ( simple+
            )>
```

D.4.1.4 *struct*.

The *struct* element is used to group properties with different characteristics (i.e., similar to a structure or record entry). Each item in the *struct* element can be a different simple type (e.g., short, long, etc.). The *struct* element corresponds to the CF *Properties* type where each *struct* item (ID, value) corresponds to a properties list item. The properties list size is based on the number of *struct* items.

```
<!ELEMENT struct
      ( description?
      , simple+
      , configurationkind?
      )>
<!ATTLIST struct
      id       ID         #REQUIRED
      name   CDATA        #IMPLIED
      mode   (readonly | readwrite | writeonly)      "readwrite">”
```

D.4.1.4.1 *configurationkind.*

The *configurationkind* element is used to specify the kind of property. The types of kinds are:

1. configure which is used in the *configure*( ) and *query* ( ) operations of the SCA *Resource* interface. The CF *ApplicationFactory* will use these properties to build the CF *Properties* input parameter to the *configure* ( ) operation that is invoked on the CF *Resource* components during application creation. When the mode is readonly, only the *query* behavior is supported. When the mode is writeonly, only the *configure* behavior is supported. When the mode is readwrite, both *configure* and *query* are supported.
2. factoryparam which is used in the *createResource* operations of the CF *ResourceFactory* interface. The CF *ApplicationFactory* will use these properties to build the CF *Properties* input parameter to the *createResource*( ) operation.”

```
<!ELEMENT configurationkind EMPTY>
<!ATTLIST configurationkind
      kindtype  (configure | factoryparam) “configure”>
```

D.4.1.5 *structsequence.*

The *structsequence* element is used to specify a list of properties with the same *struct* characteristics. The *structsequence* element maps to a *properties* element having the CF *Properties* type. Each item in the CF *Properties* type will be the same *struct* definition as referenced by the structrefid attribute.

```
<!ELEMENT structsequence
      ( description?
      , structvalue+
      , configurationkind?
      )>
<!ATTLIST structsequence
      id     ID     #REQUIRED
      structrefid CDATA #REQUIRED
      name CDATA   #IMPLIED
      mode  (readonly | readwrite | writeonly) "readwrite">

<!ELEMENT structvalue
      (simpleref+
      )>

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
      refid CDATA #REQUIRED
      value CDATA #REQUIRED>”
```

D.5 SOFTWARE COMPONENT DESCRIPTOR.

This descriptor file is based on the CORBA Component Descriptor specification. The SCA components CF *Resource*, CF *Device*, and CF *ResourceFactory* that are described by the software component descriptor are based on the SCA CF specification, and the following specification concentrates on definition of the elements necessary for describing the ports and interfaces of these components.

```
<!ELEMENT softwarecomponent

      ( corbaversion
      , componentrepid
      , componenttype
      , componentfeatures
      , interfaces
      , propertyfile?
      )>
```

D.5.1 *softwarecomponent.*

The *softwarecomponent* element is the root element to the software component descriptor file. For use within the SCA the sub-elements that are supported include:

A. *corbaversion* – indicates which version of CORBA the component is developed for.
B. *componentrepid* – is the repository id of the component
C. *componenttype* – identifies the type of software component object
D. *componentfeatures* – provides the supported message ports for the component
E. *interface* – describes the component unique id and name for supported interfaces.

D.5.1.1 *corbaversion.*

The *corbaversion* element is intended to indicate the version of CORBA that the delivered component supports.

```
<!ELEMENT corbaversion (#PCDATA)>
```

D.5.1.2 *componentrepid.*

The *componentrepid* uniquely identifies the interface that the component is implementing. The *componentrepid* may be referred to by the *componentfeatures* element. The *componentrepid* is derived from the CF *Resource*, CF *Device*, or CF *ResourceFactory*.

```
<!ELEMENT componentrepid EMPTY>
<!ATTLIST componentrepid
      repid  CDATA        #REQUIRED>
```

D.5.1.3 *componenttype.*

The *componenttype* describes properties of the component. For SCA components, the component types include resource, device, resourcefactory, domainmanager, logger, filesystem, filemanager, devicemanager, namingservice.

```
<!ELEMENT componenttype (#PCDATA)>
```

D.5.1.4 *componentfeatures.*

The *componentfeatures* element is used to describe a component with respect to the components that it inherits from, the interfaces the component supports, and its provides and uses *ports*. At a minimum, the component interface has to be a CF *Resource*, CF *ResourceFactory*, or CF *Device* interface. If a component extends the CF *Resource* or CF *Device* interface then all the inherited interfaces (e.g., CF *Resource*) are depicted as *supportsinterface* elements.

```
<!ELEMENT componentfeatures
      ( supportsinterface*
      , ports
      )>
```

D.5.1.4.1 *supportsinterface.*

The *supportsinterface* element is used to identify an IDL interface that the component supports. These interfaces are distinct interfaces that were inherited by the component's specific interface. One can widen the component's interface to be a *supportsinterface*. The repid is used to refer to the *interface* element (see *interfaces* section D.5.1.5).

```
<!ELEMENT supportsinterface EMPTY>
<!ATTLIST supportsinterface
      repid        CDATA  #REQUIRED
      supportsname CDATA  #REQUIRED>
```

D.5.1.4.2 *ports.*

The *ports* element describes what interfaces a component provides and uses. The *provides* elements are interfaces not part of the component interface but are independent interfaces known as facets (in CORBA Components terminology) (i.e. a *provides* port at the end of a path, like I/O Device or Modem Device, does not need to be a CF *Port* type). The *uses* element are CF *Port* interface types that are connected to other interfaces (*provides* or *supportinterfaces*). Any number of uses and provides elements can be given in any order. Each *ports* element has a name and references an interface by repid (see interfaces section D.5.1.5). The port names are used in the Software Assembly Descriptor for use when connecting ports together. A *ports* element also has an optional *porttype* element that allows for identification of port classification. Values for *porttype* include “data”, “control”, “responses”, and “test”. If a *porttype* is not given then control is assumed.

```
<!ELEMENT ports
      ( provides
      | uses
      )*>

<!ELEMENT provides
      ( porttype*)>
<!ATTLIST provides
      repid        CDATA  #REQUIRED
      providesname CDATA  #REQUIRED>

<!ELEMENT uses
      ( porttype*)>
<!ATTLIST uses
      repid        CDATA  #REQUIRED
      usesname     CDATA  #REQUIRED>

<!ELEMENT porttype EMPTY>
<!ATTLIST porttype
      type ( data | control | responses | test ) #REQUIRED>
```

D.5.1.5 *interfaces.*

The *interfaces* element is made up of one to many interface elements.

```
<!ELEMENT interfaces
      ( interface+
      )>
```

The *interface* element describes an interface that the component, either directly or through inheritance, provides, uses, or supports. The name attribute is the character-based non-qualified name of the interface. The repid attribute is the unique repository id of the interface which has formats specified in the CORBA specification. The repid is also used to reference an interface element elsewhere in the descriptor, for example from the *inheritsinterface* element.

```
<!ELEMENT interface
      ( inheritsinterface*)>
<!ATTLIST interface
      repid  CDATA  #REQUIRED
      name   CDATA  #REQUIRED>

<!ELEMENT inheritsinterface EMPTY>
<!ATTLIST inheritsinterface
      repid  CDATA  #REQUIRED
```

D.5.1.6 *propertyfile.*

Refer to section D.2.1.1 *propertyfile* for description and definition of *propertyfile*.

D.6 SOFTWARE ASSEMBLY DESCRIPTOR.

This section describes the XML elements of the Software Assembly Descriptor (SAD). The SAD is based on the CORBA Components Specification Component Assembly Descriptor. The intent of the software assembly is to provide the means of describing the assembled functional application and the interconnection characteristics of the SCA components within that application. The component assembly provides four basic types of application information for Domain Management. The first is partitioning information that indicates special requirements for collocation of components, the second is the assembly controller for the software assembly, the third is connection information for the various components that make up the application assembly, and the fourth is the visible ports for this assembly.

The installation of an application into the system involves the installation of a SAD file. The component assembly will either reference the components that are already installed in the target platform or causes the loading of the appropriate component *softpkg*.

The id attribute is a DCE UUID that uniquely identifies the assembly. This id is a DCE UUID value as specified in section D.2.1. The name is the user-friendly name for the CF *ApplicationFactory* name attribute.

```
<!ELEMENT softwareassembly
      ( description?
      , componentfiles
      , partitioning
      , assemblycontroller
      , connections?
      , externalports?
      )>
<!ATTLIST softwareassembly
      id     ID     #REQUIRED
      name   CDATA  #IMPLIED>
```

D.6.1 *description.*

The *description* element of the component assembly may be used to describe any information the developer would like to indicate about the assembly.

```
<!ELEMENT description (#PCDATA)>
```

D.6.2 *componentfiles.*

The *componentfiles* element is used to indicate that an assembly is made up of 1..n component files. The *componentfile* is a software package descriptor.

```
<!ELEMENT componentfiles
      ( componentfile+
      )>
```

D.6.2.1 *componentfile.*

The *componentfile* is a reference to a local file. See section D.2.1.1.1 for the definition of the *localfile* element. The *type* element is "Software Package Descriptor"

```
    <!ELEMENT componentfile
        ( localfile
        )>
  <!ATTLIST componentfile
        id     ID     #REQUIRED
        type   CDATA  #IMPLIED>
```

D.6.3 ***partitioning.***

Component partitioning specifies a deployment pattern of homes and their components to hosts. A particular usage of a component is always relative to a component home. A component instantiation is captured inside a *componentplacement*. The *hostcollocation* element allows for definition of several homes and their components to be placed on a common device. When the *componentplacement* is by itself, not inside a *hostcollocation*, it then has no collocation constraints.

```
<!ELEMENT partitioning
      ( componentplacement
      |  hostcollocation
      )*>
```

D.6.3.1 *componentplacement.*

The *componentplacement* element is used to define a particular deployment of a component home. The *componentplacement* element will be used for deploying a component through the use of a CF *ResourceFactory* in the SCA CF.

```
<!ELEMENT componentplacement
      ( componentfileref
      , componentinstantiation+
      )>
```

D.6.3.1.1 *componentfileref.*

The *componentfileref* element is used to reference a particular software component descriptor file or software package by its associated id. The refid attribute corresponds to the *componentfile* id.

```
<!ELEMENT componentfileref  EMPTY>
```

```
<!ATTLIST componentfileref
       refid  CDATA  #REQUIRED>
```

D.6.3.1.2 *componentinstantiation.*

The *componentinstantiation* element is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation* has an id attribute that is intended to identify the component and an *usagename* element that is intended for an applicable name for the instantiation. The *componentinstantiation* id may be referenced by the *usesport* and *providesport* elements within the assembly file.

The optional *componentproperties* element is a list of configure, factoryparam, and/or execparam properties values that are used in creating the component or for the initial configuration of the component.

The following sources will be searched for *componentinstantiation* element “execparam” and “readwrite and writeonly configure” properties initial values in the given precedence order:

1. The *componentproperties* element of the *componentinstantiation* element in SAD,
2. The value or default value if any from the SPD precedent property definition as stated in D.2.1.

The following sources will be searched for *componentinstantiation componentresourcefactoryref* element “factoryparam” properties initial values in the given precedence order:

A. The *resourcefactoryproperties* element of the *componentresourcefactoryref* element in the SAD,
B. The *componentproperties* element of the *componentinstantiation* element in SAD,
C. The value or default value if any from the SPD precedent property definition as stated in D.2.1.

The optional *findcomponent* element is used to obtain the CORBA object reference for the component instance. The two methods for obtaining a CORBA object reference are:

i. The *componentresourcefactoryref* element refers to a particular CF *ResourceFactory componentinstantiation* found in the assembly descriptor, which is used to obtain a CF *Resource* instance for this *componentinstantiation* element. The refid attribute refers to a unique componentinstantiation id attribute. The *componentresourcefactoryref* element contains an optional *resourcefactoryproperties* element which are the qualifiers properties for the CF *ResourceFactory create* call.

ii. The CORBA Naming Service is used to find the component CORBA object reference. The name specified in the *namingservice* element is a partial name that is used by the CF *ApplicationFactory* to form the complete context name.

The optional *findcomponent* element should be specified except when there is no CORBA object reference for the component instance (e.g., DSP code).

.

```
<!ELEMENT componentinstantiation
       ( usagename?
       , componentproperties?
       , findcomponent?
       )>
<!ATTLIST componentinstantiation
       id     ID           #REQUIRED>

<!ELEMENT usagename (#PCDATA)>

<!ELEMENT componentproperties
       ( simpleref
       | simplesequenceref
       | structref
       | structsequenceref
       )+ >

<!ELEMENT findcomponent
       ( componentresourcefactoryref
       | namingservice
       )>

<!ELEMENT componentresourcefactoryref
       ( resourcefactoryproperties?
       )>
<!ATTLIST componentresourcefactoryref
       refid  CDATA  #REQUIRED>

<!ELEMENT resourcefactoryproperties
       ( simpleref
       | simplesequenceref
       | structref
       | structsequenceref
       )+ >

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
       refid CDATA #REQUIRED
       value CDATA #REQUIRED>

<!ELEMENT simplesequenceref
       (values
       )>
<!ATTLIST simplesequenceref
       refid CDATA #REQUIRED>

<!ELEMENT structref
       (simpleref+
       )>
<!ATTLIST structref
       refid CDATA #REQUIRED>

<!ELEMENT structsequenceref
       ( structvalue+
       )>
<!ATTLIST structsequenceref
       refid CDATA #REQUIRED>

<!ELEMENT structvalue
       (simpleref+
       )>

<!ELEMENT values
       ( value+
       )>
```

```
<!ELEMENT value (#PCDATA)>
```

D.6.3.2 *hostcollocation.*

The *hostcollocation* element specifies a group of component instances that are to be deployed together in a single host. For purposes of the SCA, the *componentplacement* element will be used to describe the 1...n components that will be collocated on the same host platform. Within the SCA specification a host platform will be interpreted as a single device. The id and name attributes are optional but may be used to uniquely identify a set of collocated components within an assembly file.

```
<!ELEMENT hostcollocation
      ( componentplacement
      )+>
<!ATTLIST hostcollocation
      id     ID     #IMPLIED
      name   CDATA  #IMPLIED>
```

D.6.3.2.1 *componentplacement.*

See *componentplacement,* section D.6.3.1.

D.6.4 ***assemblycontroller.***

The *assemblycontroller* element indicates the component that is the main CF *Resource* controller for the assembly. The CF *Application* object delegates its CF *Resource start, stop, runTest* operations to this *assemblycontroller* component.

```
<!ELEMENT assemblycontroller
      ( componentinstantiationref
      )>
```

D.6.5 ***connections.***

The *connections* element is a child element of the *softwareassembly*. The *connections* element is intended to provide the connection map between components in the assembly.

```
<!ELEMENT connections
      ( connectinterface*
      )>
```

D.6.5.1 *connectinterface.*

The *connectinterface* element is used to describe the connections between the port component interfaces for the assembly of components. The *connectinterface* element consists of a *usesport* element and a *providesport, componentsupportedinterface, or findby* element. These elements are intended to connect two compatible components.

```
<!ELEMENT connectinterface
      ( usesport
      ,      ( providesport
             | componentsupportedinterface
             | findby
             )
      )>

<!ATTLIST connectinterface
      id     ID     #IMPLIED>
```

D.6.5.1.1 *usesport.*

The *usesport* element identifies, using the *usesidentifier* element, the component port that is using the provided interface from the *providesport* element. A CF *Resource* type component may be located by one of four methods. One method is to use the *componentinstantiationref* that refers to the *componentinstantiation* id (*see componentinstantiation*) within the assembly; the other techniques are *findby, devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref.*

```
<!ELEMENT usesport
      ( usesidentifier
      ,      ( componentinstantiationref
             | devicethatloadedthiscomponentref
             | deviceusedbythiscomponentref
             | findby
             )
      )>
```

D.6.5.1.1.1 *usesidentifier.*

The *usesidentifier* element identifies which uses “port” on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component ports specified in the software component descriptor.

```
<!ELEMENT usesidentifier (#PCDATA)>
```

D.6.5.1.1.2 *componentinstantiationref.*

The *componentinstantiationref* element refers to the id reference of the *componentinstantiation* within the assembly descriptor file. The refid attribute will correspond to the unique *componentinstantiation* id attribute.

```
<!ELEMENT componentinstantiationref EMPTY>
<!ATTLIST componentinstantiationref
      refid  CDATA  #REQUIRED>
```

D.6.5.1.1.3 *findby*.

The *findby* element is used to resolve a connection between two components. It tells the Domain Management function how to locate a component interface involved in the relationship of a connection. The *namingservice* element specifies a naming service name to search, and the *stringifiedobjectref* element is a stringified object reference (IOR) for the desired component interface.

The *domainfinder* element specifies an element within the domain that the Domain Management function knows about.

```
<!ELEMENT findby
      ( namingservice
      | stringifiedobjectref
      | domainfinder
      )>
```

D.6.5.1.1.3.1 namingservice.

The *namingservice* element is a child element of *findby*. The namingservice element is used to indicate to the CF *DomainManager* the requirement to find a component interface. The CF *DomainManager* will use the *name* attribute to search the CORBA naming service for the appropriate interface.

```
<!ELEMENT namingservice EMPTY
<!ATTLIST namingservice
      name   CDATA  #REQUIRED>
```

D.6.5.1.1.3.2 *stringifiedobjectref*.

The *stringifiedobjectref* element is a child element of *findby*. The *stringifiedobjectref* element is used to indicate to the CF *DomainManager* the necessary information to find a component interface by its object reference.

```
<!ELEMENT stringifiedobjectref (#PCDATA)>
```

D.6.5.1.1.3.3 *domainfinder*.

The *domainfinder* element is a child element of *findby*. The *domainfinder* element is used to indicate to the CF *ApplicationFactory* the necessary information to find an object reference that is of specific type and may also be known by an optional name within the domain. The valid types are “filemanager”, “logger”, and “namingservice”. If name is not supplied then the component reference returned is the CF *DomainManager’s FileManager*, *FileSystem*, or naming service corresponding to the type provided. If name is not supplied and the type is “log” then a null reference is returned.

```
<!ELEMENT domainfinder EMPTY>
```

```
<!ATTLIST domainfinder
      type CDATA #REQUIRED
      name CDATA #IMPLIED
```

D.6.5.1.1.4 *devicethatloadedthiscomponentref.*

The *devicethatloadedthiscomponentref* element refers to a specific component found in the assembly. This referenced component is used to obtain from the CF *ApplicationFactory* the logical CF *Device* that was used to load the referenced component. This obtained logical device is associated with this component instance. This relationship is needed when a component (e.g., modem adapter) is pushing or pulling data and/or commands to a non-CORBA capable device such as modem.

```
<!ELEMENT devicethatloadedthiscomponentref EMPTY>
<!ATTLIST devicethatloadedthiscomponentref
      refid  CDATA  #REQUIRED>
```

D.6.5.1.1.5 *deviceusedbythiscomponentref.*

The *deviceusedbythiscomponentref* element refers to a specific component found in the assembly. This referenced component is used to obtain from the CF *ApplicationFactory* the CF *Device* (e.g., logical *Device*) that is being used by this referenced component. This relationship is needed when a component is pushing or pulling data and/or commands to another component that exists in the system such as an audio device.

```
<!ELEMENT deviceusedbythiscomponentref EMPTY>
<!ATTLIST deviceusedbythiscomponentref
      refid         CDATA    #REQUIRED
      usesrefid   CDATA    #REQUIRED>
```

D.6.5.1.2 *providesport.*

The *providesport* element identifies, using the *providesidentifier* element, the component port that is providing to the *usesport* interface within the *connectinterface* element. A CF *Resource* type component may be located by one of four methods. One method is to use the *componentinstantiationref* that refers to the componentinstantiation id (*see componentinstantiation*) within the assembly; the other techniques are *findby, devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref*. The *findby* element by iteself is used when the object reference is not a CF *Resource* type.

```
<!ELEMENT providesport
      ( providesidentifier
      , ( componentinstantiationref
        | devicethatloadedthiscomponentref
        | deviceusedbythiscomponentref
        | findby
        )
      )>
```

D.6.5.1.2.1 *providesidentifier.*

The *providesidentifier* element identifies which provides "port" on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component ports specified in the software component descriptor.

```
<!ELEMENT providesidentifier (#PCDATA)>
```

D.6.5.1.2.2 *componentinstantiationref.*

See D.6.5.1.1.2 for a description of the *componentinstantiationref* element.

D.6.5.1.2.3 *findby*.

See section D.6.5.1.1.3 for a description of the *findby* element. The namingservice name denotes a complete naming context.

D.6.5.1.2.4 *devicethatloadedthiscomponentref.*

See D.6.5.1.1.4 for a description of the *devicethatloadedthiscomponentref* element.

D.6.5.1.2.5 *deviceusedbythiscomponentref.*

See D.6.5.1.1.5 for a description of the *deviceusedbythiscomponentref* element.

D.6.5.1.3 *componentssupportedinterface.*

Specifies a component with a *supports* interface that can satisfy an interface connection to a *uses* port within a *connectinterface* element. The component is identified by a *componentinstantiationref* or a *findby* element. The *componentinstantiationref* identifies a component within the assembly. The *findby* element points to an existing component that can be found within a naming service or using a stringified object reference.

```
<!ELEMENT componentsupportedinterface
      ( supportedidentifier
      , ( componentinstantiationref
        | findby
        )
      )>
```

D.6.5.1.3.1 *supportedidentifier*.

The *supportedidentifier* element identifies which supported interface on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component supported interface specified in the software component descriptor.

```
<!ELEMENT supportedidentifier (#PCDATA)>
```

D.6.5.1.3.2 *componentinstantiationref.*

See section D.6.5.1.1.2 for a description of the *componentinstantiationref* element.

D.6.5.1.3.3 findby.

See section D.6.5.1.1.3 for a description of the *findby* element.

D.6.6 *externalports.*

The optional *externalports* element is a child element of the *softwareassembly*. The *externalports* element is used to identify the visible ports for the software assembly. The CF *Application getport* ( ) operation is used to access the assembly visible ports.

```
<!ELEMENT externalports
      ( port+
      )>

<!ELEMENT port
      ( description?
      , ( usesidentifier | providesidentifier |
          supportedidentifier)
      , componentinstantiationref
      )>
<!ELEMENT description (#PCDATA)>
```

D.7 DEVICE CONFIGURATION DESCRIPTOR.

This section describes the XML elements of the Device Configuration Descriptor (DCD). The DCD is based on the SAD (e.g., componentfiles, partitioning, etc.) DTD. The intent of the DCD is to provide the means of describing the components that are initially started on the CF *DeviceManager* node, how to obtain the CF *DomainManager* object reference, connections of services to components (CF *Device*s, CF *DeviceManager*), and the characteristics (file system names, etc.) for a CF *DeviceManager*. The *componentfiles* and *partitioning* elements are optional; if not provided, that means no components are started up on the node, except for a CF *DeviceManager*. If the *partitioning* element is specified then a *componentfiles* element has to be specified also.

The id attribute is a unique identifier within the domain for the device configuration. This id attribute can be a UUID value as specified in section D.2.1 or a manufacturer's part number – serial number string, for example. The name attribute is the user-friendly name for the CF *DeviceManager's* label attribute.

```
<!ELEMENT deviceconfiguration
            ( description?
            , devicemanagersoftpkg
            , componentfiles?
            , partitioning?
            , connections?
            , domainmanager
            , filesystemnames?
            )><!ATTLIST deviceconfiguration
      id     ID      #REQUIRED
      name   CDATA   #IMPLIED>
```

D.7.1 *description.*

The optional *description* element of the device configuration may be used to provide information about the device configuration.

```
<!ELEMENT description (#PCDATA)>
```

D.7.2 *devicemanagersoftpkg.*

The *devicemanagersoftpkg* element refers to the SPD for the CF *DeviceManager* that corresponds to this DCD. The SPD file is referenced by a *localfile* element. This SPD can be used to describe the CF *DeviceManager* implementation and to specify the *usesports* for the services (Log*(s)*, etc.) used by the CF *DeviceManager*. See (section D.2.1.1.1) for description of the *localfile* element.

```
<!ELEMENT devicemanagersoftpkg
      ( localfile
      ) >
```

D.7.3 *componentfiles.*

The optional *componentfiles* element is used to indicate that a node uses 1..n component files to define the components that are started up on the node. The componentfile is a software package descriptor. The SPD for example, can be used to describe logical *Devices*, CF *DeviceManager*, CF *DomainManager*, naming service, and CF *FileSystems*. Refer to section D.6.2 for *componentfiles* definition.

D.7.4 *partitioning.*

The optional *partitioning* element consists of a set of *componentplacement* elements. A component instantiation is captured inside a *componentplacement*.

```
<!ELEMENT partitioning
      ( componentplacement
)*>
```

D.7.4.1 *componentplacement.*

The *componentplacement* element is used to define a particular deployment of a component. The *componentfileref* element identifies the component to be deployed. The *componentinstantiation* element identifies the actual component created and its id attribute is a DCE UUID value with the format as specified in section D.2.1. Multiple components of the same kind can be created within the same *componentplacement* element.

The optional *deployondevice* element indicates the device the *componentinstantiation* element is deployed on. The optional *compositepartofdevice* element indicates the device the *componentinstantiation* element is an aggregate of (aggregation relationship). When the component is a logical *Device*, the devicepkgfile element is specified to indicate the hardware device information for the logical *Device*.

```
<!ELEMENT componentplacement
      ( componentfileref
      , deployondevice?
      , compositepartofdevice?
      , devicepkgfile?
      , componentinstantiation+
      )>
```

D.7.4.1.1 *componentfileref.*

The *componentfileref* element is used to reference a component file within the *componentfiles* element. The refid attribute corresponds to the *componentfile* id.

```
<!ELEMENT componentfileref  EMPTY>
<!ATTLIST componentfileref
      refid  CDATA  #REQUIRED>
```

D.7.4.1.2 *deployondevice*.

The *deployondevice* element is used to reference a *componentinstantiation* element on which this *componentinstantiation* is deployed.

```
<!ELEMENT deployondevice  EMPTY>
<!ATTLIST deployondevice
      refid  CDATA  #REQUIRED>
```

D.7.4.1.3 *devicepkgfile.*

The *devicepkgfile* element is used to refer to a device package file that contains the hardware device definition.

```
<!ELEMENT devicepkgfile
      ( localfile
      )>
<!ATTLIST devicepkgfile
      type    CDATA    #IMPLIED>
```

D.7.4.1.3.1 *localfile.*

See D.2.1.1.1 for a definition of the *localfile* element.

D.7.4.1.4 *compositepartofdevice*.

The *compositepartofdevice* element is used to reference a *componentinstantiation* element of which that this *componentinstantiation* element is a part (aggregate relationship).

```
<!ELEMENT compositepartofdevice  EMPTY>
<!ATTLIST compositepartofdevice
      refid  CDATA  #REQUIRED>
```

D.7.4.1.5 *componentinstantiation.*

The *componentinstantiation* element is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation* element has an id attribute that is intended to identify the component and a *usagename* element that is intended for an applicable name for the component. The optional *componentproperties* element is a list of property values that are used in configuring the component. D.6.3.1.2 has the property list for the *componentinstantiation* element initial properties values. For a component service type (e.g, Log), the *usagename* element needs to be unique for that service type.

```
<!ELEMENT componentinstantiation
      ( usagename?
      ,componentproperties?
      )>
<!ATTLIST componentinstantiation
      id    ID           #REQUIRED>
```

```
<!ELEMENT usagename (#PCDATA)>

<!ELEMENT componentproperties
( simpleref
      | simplesequenceref
      | structref
      | structsequenceref
      )+ >

<!ELEMENT simpleref EMPTY><!ATTLIST simpleref
      refid CDATA #REQUIRED
      value CDATA #REQUIRED>

<!ELEMENT simplesequenceref
      ( values
      )>
<!ATTLIST simplesequenceref
      refid CDATA #REQUIRED>

<!ELEMENT structref
      ( simpleref+
      )>
<!ATTLIST structref
      refid CDATA #REQUIRED>

<!ELEMENT structsequenceref
      ( structvalue+
      )>
<!ATTLIST structsequenceref
      refid CDATA #REQUIRED>
<!ELEMENT structvalue
      ( simpleref+
      )>

<!ELEMENT values
      ( value+
      )>

<!ELEMENT value (#PCDATA)>
```

D.7.5 *connections.*

The *connections* element in the DCD is the same as the *connections* element in the SAD in section D.6.5. The *connections* element in the DCD is used to indicate the services (Log, etc…) instances that are used by the CF *DeviceManager* and CF *Device* components in the DCD. The CF *DomainManager* will parse the connections element and make the connections when the CF *DeviceManager* registers with the CF *DomainManager*. To establish connections to a CF *DeviceManager* the DCD's id attribute value is used for the *usesport componentinstantiationref* refid value.

D.7.6 *domainmanager.*

The *domainmanager* element indicates how to obtain the CF *DomainManager* object reference.

See sections D.6.5.1.1.3.1 and D.6.5.1.1.3.2 for description of the *namingservice* and *stringifiedobjectref* elements.

```
<!ELEMENT domainmanager
. ( namingservice
      | stringifiedobjectref
      )>


<!ELEMENT namingservice EMPTY>
<!ATTLIST namingservice
      name      CDATA      #REQUIRED>

<!ELEMENT stringifiedobjectref (#PCDATA)>
```

D.7.7 *filesystemnames.*

The optional *filesystemnames* element indicates the mounted file system names for CF *DeviceManager's FileManager*.

```
<!ELEMENT filesystemnames
      ( filesystemname+
      )+

<!ELEMENT filesystemname EMPTY>
<!ATTLIST filesystemname
      mountname  CDATA  #REQUIRED
      deviceid   CDATA  #REQUIRED>
```

D.8 *DOMAINMANAGER* CONFIGURATION DESCRIPTOR.

This section describes the XML elements of the *DomainManager* Configuration Descriptor (DMD). The DMD is used to describe the *CF DomainManager*.

```
<!ELEMENT domainmanagerconfiguration
      ( description?
      , domainmanagersoftpkg
      , services
      )>
<!ATTLIST domainmanagerconfiguration
      id    ID    #required
      name #CDATA #required>
```

D.8.1 *description*.

The optional *description* element of the DMD may be used to provide information about the configuration.

```
<!ELEMENT description (#PCDATA)>
```

D.8.2 *domainmanagersoftpkg*.

The *domainmanagersoftpkg* element refers to the SPD for the CF *DomainManager*. The SPD file is referenced by a *localfile* element. This SPD can be used to describe the CF *DomainManager* implementation and to specify the *usesports* for the services (Log*(s)*, etc…) used by the CF *DomainManager*. See section D.2.1.1.1 for description of the localfile element.

```
<!ELEMENT domainmanagersoftpkg
( localfile
) >
```

D.8.3 *services*.

The *services* element in the DMD is used by the CF *DomainManager* to determine which service (Log, etc…) instances to use. See section D.6.5.1.1.3 for a description of the *findby* element.

```
<!ELEMENT services
      ( service+
      ) >
<!ELEMENT service
      ( usesidentifier
      , findby
      )>
```

D.9 PROFILE DESCRIPTOR.

The *profile* element can be used to specify the absolute profile file pathname relative to a mounted CF *FileSystem*. The filename attribute is the absolute pathname relative to a mounted

FileSystem. This filename can also be used to access any other local file elements in the profile. The type attribute indicates the type of profile being referenced. The valid type attribute values are “SAD”, “SPD”, “DCD”, and “DMD”. This element can be given out for any CF interface (e.g., CF *Application*, CF *Device*, CF *ApplicationFactory,* CF *DeviceManager, CF DomainManager*) that has profile attributes.

```
<!ELEMENT profile EMPTY>
<!ATTLIST profile
      filename     CDATA #REQUIRED
      type         CDATA #IMPLIED>
```

D.10 DOCUMENT TYPE DEFINITIONS.

Attachment 1 to Appendix D contains the complete DTDs for the Domain Profile. This text file of the DTDs is available electronically.